Génie Logiciel Avancé
Cours 3 — Le modèle à objets

Stefano Zacchiroli
zack@pps.univ-paris-diderot.fr

Laboratoire PPS, Université Paris Diderot - Paris 7

URL http://upsilon.cc/zack/teaching/1112/gla/



Sommaire

Sommaire

Le modèle à objets

- Lorsque l'on développe un système ayant une contrepartie physique, une association de la forme «1 *objet physique* ↔ *1 composant logiciel*» peut être tentante.
- Dans le cours de programmation objet vous avez vu le «triangle sémiotique» : les analogies «référent»/«instance» et «signifié»/«interface» peuvent faciliter le raisonnement et, surtout, la validation d'une spécification vis-à-vis des besoins
  - Est-ce que je construis le bon logiciel ?
- Cette correspondance facilite la discussion avec un non-expert
  - on peut utiliser un composant logiciel par son nom devant un client non informaticien et celui-ci peut comprendre à peu près de quoi il retourne.

Note

Ce cours suppose quelques connaissances en programmation orientée objet.

Principes généraux de GL... dans le modèle à objets

Définition (Objet)

Un objet est formé d'un état et d'un ensemble de comportements modélisés comme des réactions à des messages.
Il a une identité. Sa durée de vie est limitée. Il joue un ou plusieurs rôles dans le système.

Principes :

Modularité La logique interne de l'objet est décorrélée de son utilisation.

Encapsulation La seule façon d'influer sur l'état d'un objet est de lui envoyer des messages.

Abstraction Les objets sont généralement classifiés suivant une relation de généralisation.

Les forces du modèle à objets

- En plus des apports mentionnés plus tôt, les objets facilitent un raffinement progressif du modèle logique à l'implémentation.
- En effet, les concepts importants du système sont souvent modélisés par des classes abstraites dont les sous-classes fournissent des concrétisations.
- De plus, les objets améliorent la réutilisabilité grâce à leur relative indépendance vis-à-vis du contexte d'utilisation.
- Enfin, l'extension *a posteriori* d'un composant est autorisée par le mécanisme d'héritage.
  - Cette extension n'est pas intrusive : elle ne nécessite pas de reprendre à zéro le raisonnement sur le système dans sa globalité (*separation of concerns*).

Les faiblesses du modèle à objets

Malgré son utilisation très répandue, le modèle à objet n'est pas la solution ultime aux problèmes de la définition de composants logiciel réutilisables, corrects et robustes.

Faiblesses du modèle à objets

1. La non-transparence observationnelle : à cause de son état interne, la réaction d'un objet à un message n'est pas toujours la même. Ceci rend difficile le raisonnement sur les objets.
2. Le mécanisme d'héritage ne reflète pas la même intention en fonction du niveau d'abstraction auquel on se place. En effet, dans un modèle logique, l'héritage sert à refléter la relation de généralisation/spécialisation. Plus on se rapproche d'une spécification technique et plus cette relation est un mécanisme de réutilisation de code.

⇒ Le cours de POCA de Master 2.

Les faiblesses du modèle à objets (cont.)

Malgré son utilisation très répandue, le modèle à objet n'est pas la solution ultime aux problèmes de la définition de composants logiciel réutilisables, corrects et robustes.

Faiblesses du modèle à objets (cont.)

3. Les opérations *n*-aires sont peu compatibles avec le modèle objet (p.ex. le problème du *multiple dispatching*)
4. La notion de message n'est pas de première classe, ce qui rend compliquée l'expression de mécanismes calculatoires de la forme «pour tout message, ...» (solution partielle : *aspect oriented programming*).
5. On aimerait parfois raisonner sur le système comme un monde clos en interdisant certaines extensions futures dangereuses (solution partielle : *final classes*).

⇒ Le cours de POCA de Master 2.

Sommaire

Le Rational Unified Process (RUP)

Le Rational Unified Process (RUP) développé par IBM est une famille de processus.

- Ce sont des processus itératifs incrémentaux centrés sur le modèle à objets et sur UML (voir plus loin dans ce cours)
- Les validations de chaque phase s'appuient sur des cas d'utilisation.
- Le système est décrit comme la somme de multiples vues.
- Son architecture est le soucis permanent : le RUP préconise le développement préliminaire d'une architecture exécutable, c'est-à-dire une version du système avec un nombre très limité de fonctionnalités mais dont le "squelette" est fixé.

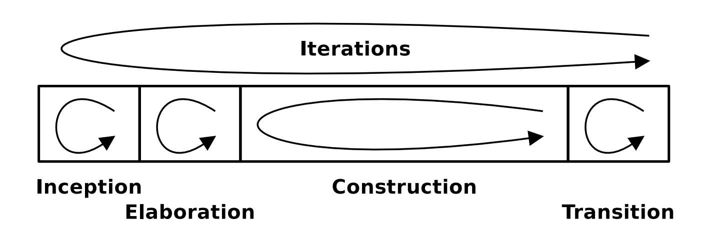
Les différentes variantes du RUP

- Unified Process (UP) est la version la plus documentée du RUP
  - C'est une version générique adaptable aux besoins particuliers.
- Agile Unified Process (AUP) ajoute un caractère *évolutif* à RUP
  - On s'appuie sur une haute qualification des développeurs pour limiter le plus possible la production de documents préliminaires au développement. Les cas d'utilisation sont représentés par des tests exécutables. Un prototype est développé très rapidement et dirigés par la validation de ces tests. La spécification est construite en même temps que le logiciel, une fois que celui-ci est confronté sous une forme exécutable aux utilisations des clients.

Perspectives du RUP

1. Perspective dynamique
   - les phases du processus et leur ordre temporel
2. Perspective statique
   - les *work-flows* qui correspondent aux activités de développeurs et d'autre acteurs
3. Perspective pratique
   - les bonnes pratiques (*best practices*) à utiliser pendant toute la durée du processus

Perspective dynamique — les phases du RUP



1. initialisation (*inception*)
2. élaboration
3. construction
4. transition

Itération

- chaque phase peut être itérée plusieurs fois avant de passer à la phase successive (micro-itération)
- l'ensemble de phases est typiquement itéré plusieurs fois, comme dans tous les modèle itératifs (macro-itération)

Phase initialisation (*inception*)

Cette phase correspond à l'étude de faisabilité dont nous avons parlée dans le cours précédent.

- établir un *business case* pour le système
- il peut amener à l'abandon du projet (donc il est plus intense vers le début du projet, pour minimiser les risques)

Phase élaboration

Il s'agit de l'analyse des besoins.

- Celle-ci fait un usage intensif des cas d'utilisation (et donc de scénarios) pour raffiner la compréhension du problème posé et expliciter les spécificités du domaine.
- Des prototypes (parmi lesquels on trouve l'architecture exécutable) sont développés pour évaluer concrètement des points techniques risqués.

Phase construction

Cette phase correspond à la conception et au développement.

- Elle est répète plusieurs fois pour une progression incrémentale aboutissant à diverses versions du système, résolvant les problèmes techniques à hauts risques en priorité.
- Dans une conception orienté objet, il est parfois difficile de bien distinguer la spécification de l'implémentation.
  - Certains spécialistes préconisent la définition de deux modèles disjoints : un modèle logique et un modèle d'implémentation (voir : *model-driven engineering*).
  - Cette distinction est importante car il doit toujours exister une spécification servant de référence aux développements et sur laquelle appuyé le cahier des charges.
  - À partir des modèles logiques et d'implémentation, *model-driven engineering* préconise la génération de beaucoup de code d'implémentation

Phase transition

- La phase de transition de ce processus correspond à l'activité de déploiement et marque le début de la maintenance du logiciel
- Il s'agit de vérifier la mise en place du système auprès des utilisateurs (production de manuel d'utilisation, formation, . . .) et de préparer ses futures évolutions.
- Cette phase a été ignoré par plusieurs modèles de développement précédents au RUP.

Perspective statique — les *work-flows* du RUP

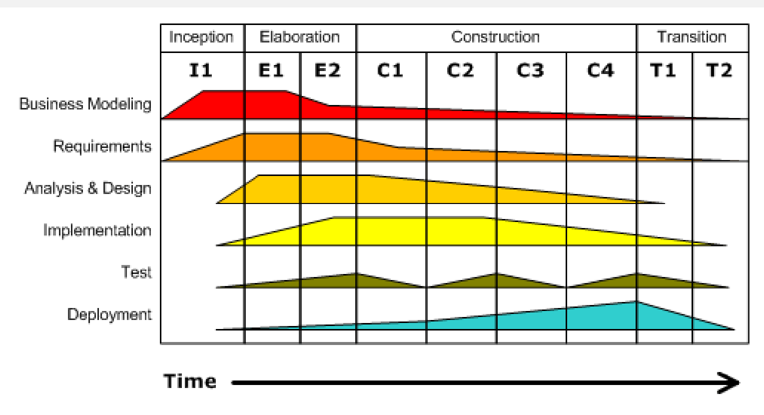
Core *work-flows*

- *business modeling*
  - i.e. modélisation du contexte du logiciel, point de vue business
- *requirements*
- *analysis and design*
- *implementation*
- *testing*
- *deployment*

Support *work-flows*

- *configuration and change management*
- *project management*
- *environment*
  - i.e. gestion des outils (de développement ou autres) nécessaires dans les différentes phases et *work-flows*

Interaction entre phases et *work-flows*



http://en.wikipedia.org/wiki/File:Development-iterative.gif

- les lignes correspondent aux *work-flows*
- les colonnes correspondent aux phases
- la hauteur détermine l'intensité de *work-flows* dans chaque micro-itération
- on répète tout le diagramme pour chaque macro-itération

Perspective pratique

Bonnes pratiques encouragées par RUP :

1. développement itératif
2. gestion explicite de requis
3. utilisation de architecture à composante
4. modélisation semi-formelle et visuelle du logiciel avec UML
5. assurance qualité
6. gestion du changement (logiciel et spécification)

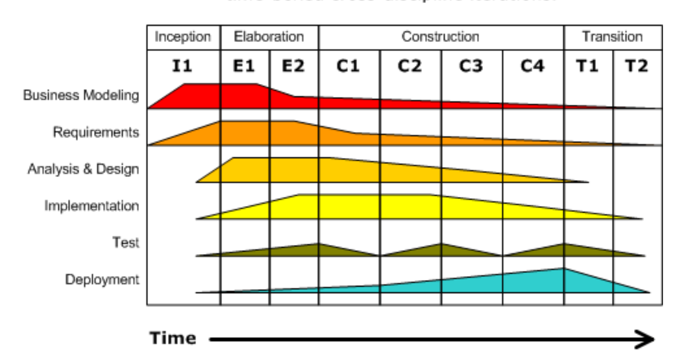
Commentaire :

- (3), (5), et (6) sont encouragée explicitement dans RUP, mais ne sont pas liée au reste du processus
- l'importante du (4) est discutable et à comparer avec autres langages des modélisation logiciel

Innovations du RUP

- Séparation entre phases et *work-flows*
  - chaque *work-flow* est inter-phase, comme dans la réalité du projets de développement
- *Work-flow* explicite pour le deployment du logiciel

Cours d'aujourd'hui



- Nous allons nous intéresser à l'utilisation d'UML dans ces différentes phases pour les trois premières itérations.
  - Dans ce processus, elles correspondent à la spécification du système.

Sommaire

Présentation d'UML

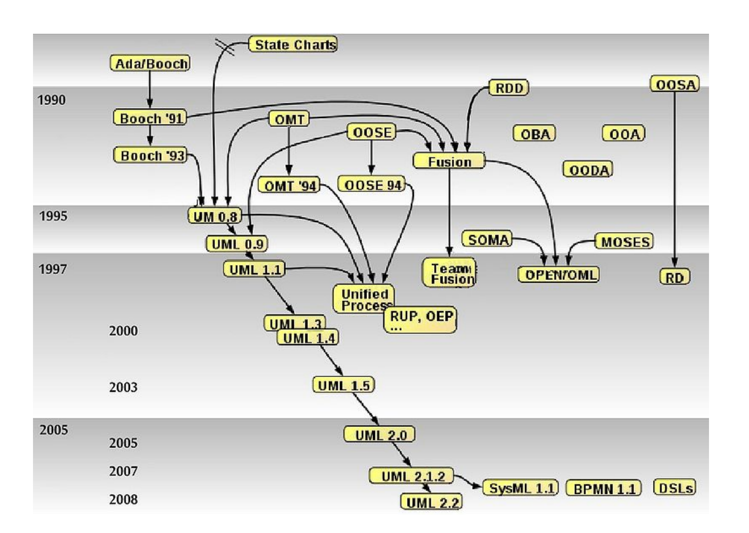
- UML est l'acronyme de *Unified Modeling Language*.
- UML est un ensemble de notations.
  - Ces notations sont en majorité des formats de diagrammes.
- UML est standardisé par l'*Object Management Group* (OMG).
- UML est la notation la plus utilisée par l'industrie logicielle.
- La dernière version de la spécification d'UML est toujours disponible à :

  `http://www.omg.org/spec/UML/Current`

  version courante : 2.4.1, Août 2011

  - Nous ne pourrons pas l'étudier en détails.
  - Vous devez vous y référer pour écrire vos spécifications.

Histoire



- Résolution des conflits par union plutôt que par intersection.

Critiques d'UML

Avantages

- Plusieurs modèles sont réunis : objet, orienté donnée, flot de données.
- Il existe de nombreux outils pour produire des diagrammes UML.
- C'est le résultat d'un consensus entre plusieurs «écoles» de modélisation.

Inconvénients

- La sémantique d'UML n'est pas encore fixée.
  - Toutefois, des experts essaient de définir *Precise UML*, un sous-ensemble formalisé d'UML.
- C'est seulement depuis la version 2.0 que la syntaxe est standardisée.
- Les notations sont parfois redondantes.



Différentes vues sur un système

UML fournit des diagrammes pour plusieurs types de vues sur un logiciel.

Dans ce cours on en regardera 4 :

1. les vues de cas d'utilisation ; [1]
2. les vues d'architecture ;
3. les vues dynamiques ;
4. les vues statiques.

1. *A priori*, il s'agit d'une vue dynamique

Une abondance de diagrammes ...

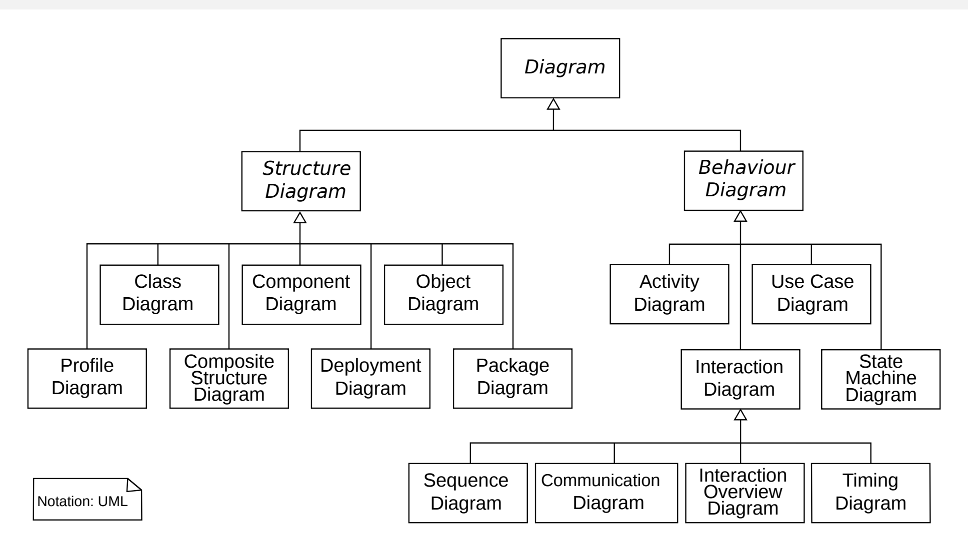


Figure: Les diagrammes d'UML 2.x

http://en.wikipedia.org/wiki/File:UML_diagrams_overview.svg

Sommaire

Les cas d'utilisation

Définition (Cas d'utilisation (rappel))

Un cas d'utilisation est la représentation d'une interaction entre le système et un acteur en vue de la réalisation d'un objectif.

- On applique ici le principe de *separation of concerns*
  - on se focalise sur une certaine utilisation du système en oubliant le reste.
- En plus de réduire temporairement la complexité du système, cette unité de description est intéressante car elle est accessible aux clients non experts.
- Lorsque l'on suit RUP, les cas d'utilisation sont décrits par deux notations :
  - les spécifications en langage structuré (vues la dernière fois)
  - les diagrammes de cas d'utilisation d'UML
- La méthode *Agile* représente les cas d'utilisation par des *programmes exécutables* pour pouvoir vérifier leur satisfiabilité de automatiquement et quantifier l'avancée du développement.

Retour sur l'exemple en langage structuré

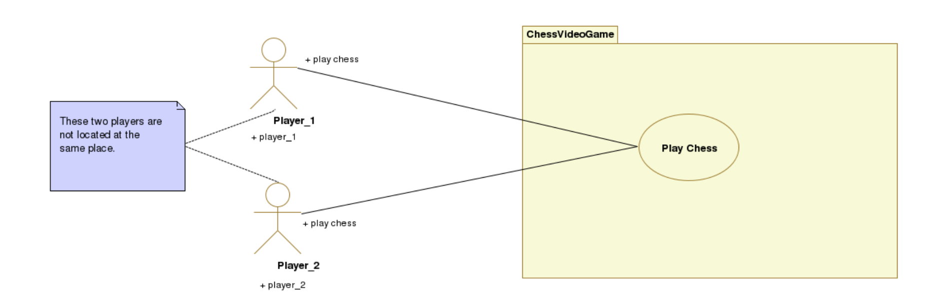
Contexte	Une partie est en cours. Le joueur a formulé une requête d'action.
Flot normal	La requête est excaucée. L'état de la partie est modifiée en accord avec le scénario et l'interface graphique est mise-à-jour. Un message (textuel ?) informe le joueur du changement.
Cas problématique	L'action n'est pas applicable. Le joueur est informé des causes de l'erreur. Il peut formuler une autre action.
Activités concurrentes	Les animations de la scène se poursuivent tout au long de la résolution de la requête d'action.

Table: scénario «résolution d'une action».

“Grammaire” des diagrammes de cas d’utilisation

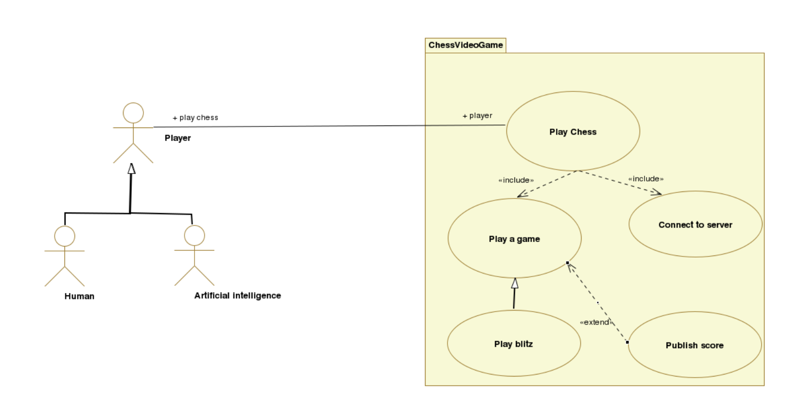
- On représente un acteur par un personnage schématisé.
  - Attention, cependant, un acteur n’est pas forcément un utilisateur ! (autres systèmes, API, etc.)
- Le système est inclus dans un rectangle (sa frontière), éventuellement étiqueté.
- Les interactions entre le système et les acteurs sont représentées par des lignes.
- Les cas d’utilisation sont des verbes à l’infinitif entourés par des ellipses.
- On représente des relations logiques (voir plus loin) :
  1. entre les acteurs
  2. entre le cas d’utilisation

Exemple : cas d'utilisation



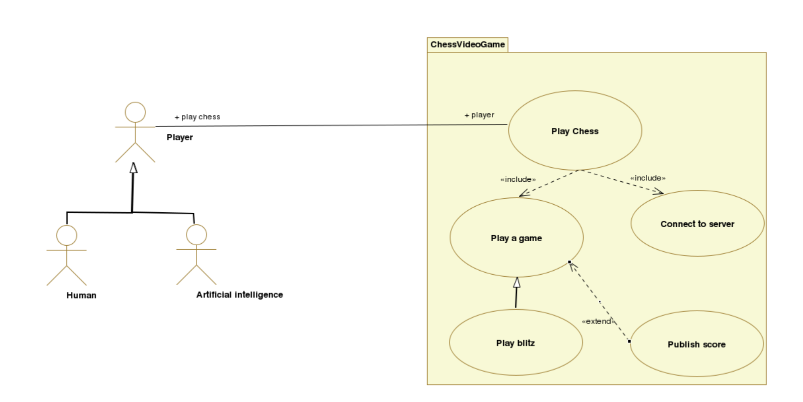
- En général, tout diagramme UML peut être annoté par un complément textuel d'information attaché à ces entités visuelles.

Exemple : cas d'utilisation (cont.)



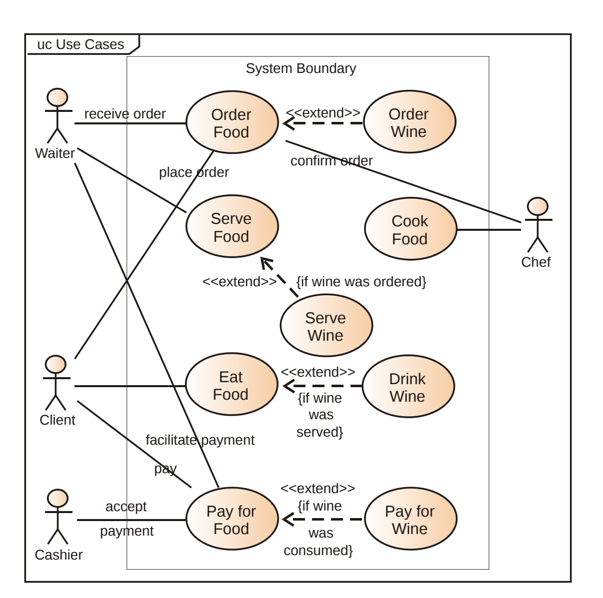
- Une relation d'héritage permet de classifier les acteurs.
- Si l'acteur $\mathcal{A}_2$ hérite de l'acteur $\mathcal{A}_1$ alors tous les scénarii de $\mathcal{A}_1$ sont accessibles à $\mathcal{A}_2$.

Exemple : cas d'utilisation (cont.)



- On retrouve ici les relations «étend» et «inclus» définies la dernière fois.

Exemple : cas d'utilisation (cont.)



http://en.wikipedia.org/wiki/File:Use_case_restaurant_model.svg

Étude de cas : diagramme de cas d'utilisation

Exercice

Traduisez la description textuelle du cas d'utilisation en un diagramme.

Rôle des cas d'utilisation dans RUP

Ils jouent un rôle central.

1. Ils servent de matériau de base à la phase de élaboration.
   - Classifier les cas d'utilisation, en termes logiques, de priorité ou de risque, permet d'organiser l'analyse qui suit.
2. De nombreuses vues dynamiques sont des raffinements des cas d'utilisation.
   - Ces raffinements précisent le vocabulaire et les mécanismes mis en jeu.
3. Si une nouvelle utilisation du système apparaît pendant l'élaboration (i.e. l'analyse de besoins), il est systématiquement inséré dans la base des cas d'utilisation.
4. La partie validation de la phase transition consiste souvent à formuler une version vérifiable/exécutable des cas d'utilisation et à y confronter le système.
5. Enfin, le manuel d'utilisation du système s'appuie très largement sur cette base de connaissance.

Activités liées à l'explicitation des cas d'utilisation

À titre indicatif, voici une succession d'activités pouvant mener à l'obtention des cas d'utilisation :

1. Identification des acteurs principaux.
   - Les acteurs à satisfaire en priorité.
   - Les entités externes vitales au système.
2. Identification des cas d'utilisation principaux.
   - On omet les situations exceptionnelles.
   - On obtient une description intentionnelle (centrée sur les objectifs).
   - On met à jour les termes et concepts incontournables du système.

Activités liées à l’explicitation des cas d’utilisation

À titre indicatif, voici une succession d’activités pouvant mener à l’obtention des cas d’utilisation :

3. Identification des acteurs secondaires.
   - Des acteurs qui interviennent dans les cas d’utilisation découverts.
4. Identification des cas d’utilisations secondaires.
   - Par raffinement des cas d’utilisation principaux.
5. Factorisation des redondances.
6. Définition du vocabulaire du domaine.
   - Les cas d’utilisation soulèvent des questions sur le sens des termes employés par les acteurs.

Étude de cas : extraction des cas d’utilisation

Exercice

Définissez les acteurs et cas d’utilisation principaux du moteur générique de jeu d’aventure.

Critique des cas d'utilisation

Malgré les qualités citées plus tôt, la centralisation autour des cas d'utilisation peut avoir des faiblesses :

Taille l'énumération des cas d'utilisation et de leurs variations peut induire une combinatoire assez importante.

Conception le point de vue «utilisateur» n'est pas forcément la bonne façon d'aborder un problème.

- Par exemple, les utilisateurs peuvent avoir une vue incomplète du problème ou être des instances (inconscientes) de problèmes plus généraux.

Imprécision il est très difficile d'avoir un discours précis en s'exprimant seulement à l'aide de cas d'utilisation.

- Formaliser rapidement les concepts ou processus primordiaux permet d'en saisir les subtilités.

Sommaire

Vues d'architecture

- L'architecture est une vue d'ensemble du système.
- C'est un point de conception à haut risque.
- Il s'agit de partitionner le système en sous-systèmes.

- Un bon partitionnement établit :
  - une faible dépendance entre les sous-systèmes ;
  - affecte un rôle clair et distinct à chaque sous-système ;
  - permet de couvrir l'ensemble des les cas d'utilisation.

Exemple : paquets

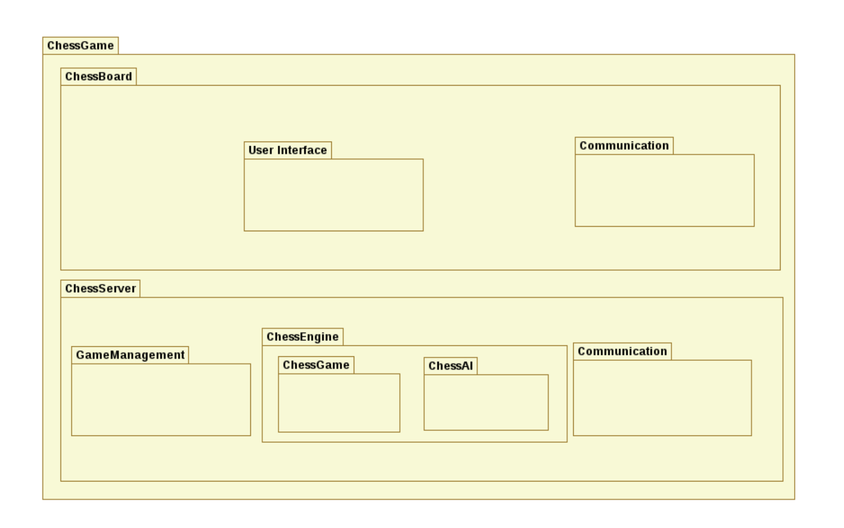


Figure: Diagramme d'architecture (ou "de paquets").

Diagramme des paquets : contenu

Un diagramme des paquets peut contenir :

1. classes importantes du système, pour associer classes aux sous-systèmes
2. cas d'utilisation, pour montrer les cas d'utilisation gérés par un sous-système spécifiques

Diagramme des paquets : relations

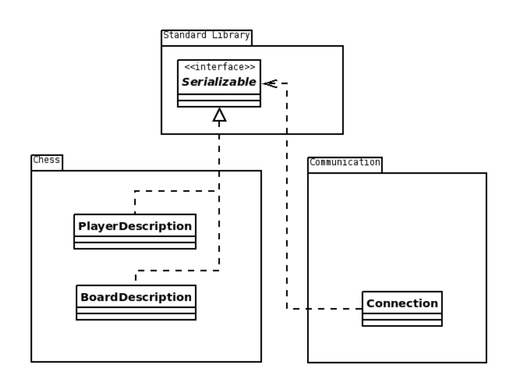
Les relations suivantes peuvent être définies entre sous-systèmes dans des diagrammes des paquets :

dépendance l'"utilisation" (vague. . .) d'un sous-système par un autre

import *a relationship between an importing namespace and a package, indicating that the importing namespace adds the names of the members of the package to its own namespace*

merge *a directed relationship between two packages, that indicates that the contents of the two packages are to be combined.*

Exemple : paquets avec classes et dépendances

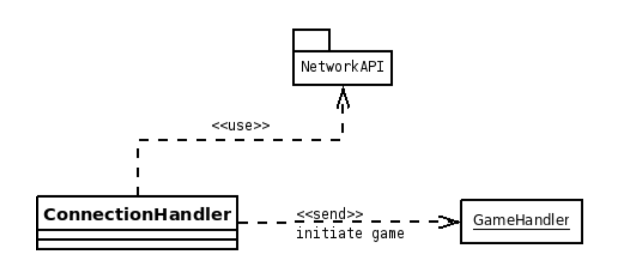


- Les «boîtes» qui apparaissent ici sont des classes importantes du système que l'on commence à classifier en termes de leur appartenance à un sous-système donné.
- Les relations entre ces classes (voir plus loin) induisent des dépendances entre les sous-systèmes.

Effets de dépendances

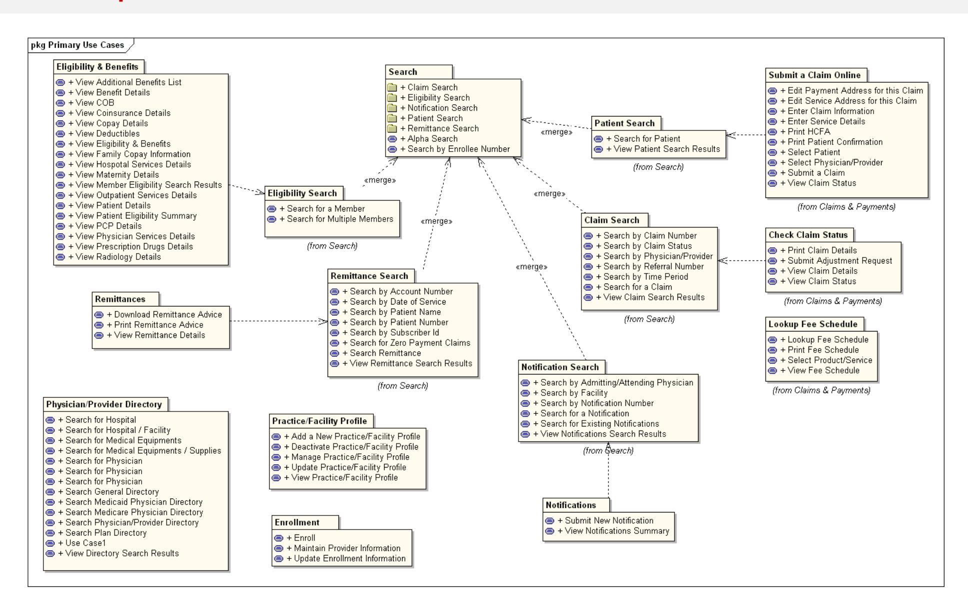
- Lorsqu'un sous-système dépend d'un autre, on doit commencer par établir l'interface de ce dernier.

Exemple : paquets avec classes et annotations



- On peut préciser les relations de dépendances à l'aide d'annotations.
  - Voir la spécification d'UML pour connaître les annotations standards.

Exemple : avec cas d'utilisations et relations



http://en.wikipedia.org/wiki/File:Package_Diagram.PNG

Diagramme de déploiement

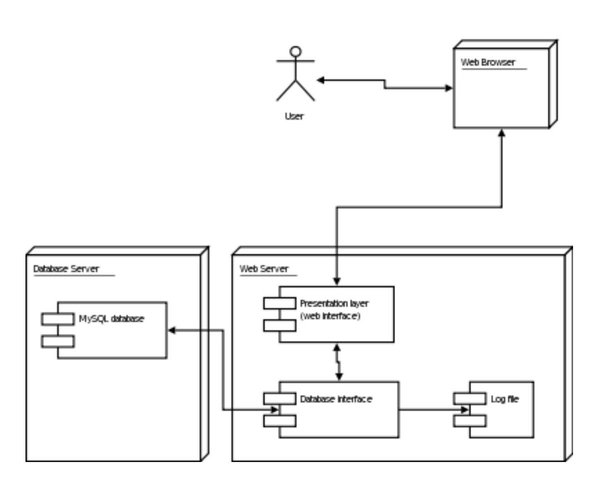
Définition (Diagramme de déploiement)

Un diagramme de déploiement (*deployment diagram*) montre le plan de déploiement d'un système, quand le système sera complet.

Dans un diagramme de déploiement, une association (*mapping*) entre artefacts et noeuds est établie

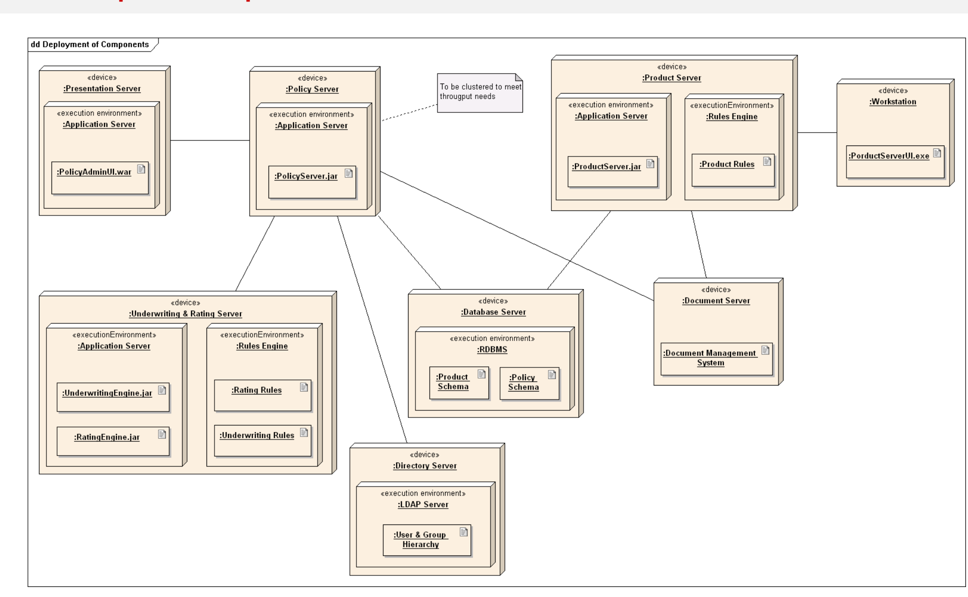
- les artefacts sont des entités "physiques" produites ou utilisées par le processus de développement logiciel (p ex. fichiers, code objets, bases de donnes, documents, etc.)
- les noeuds sont des ressources de computation, sur lesquelles les artefacts peuvent être déployés (p. ex. serveurs matériels), ou des environnements d'exécution (p. ex. serveurs logiciels, *framework*, etc.)
  - en général, les noeuds sont organisés dans une hiérarchie de noeuds

Exemple : déploiement



http://en.wikipedia.org/wiki/File:UML_Diagramme_Deploiement.gif

Exemple : déploiement



http://en.wikipedia.org/wiki/File:Deployment_Diagram.PNG

Sommaire

Vues dynamiques

- Les vues dynamiques décrivent le comportement du système (*behaviour*).
- Elles permettent de
  1. préciser les cas d'utilisation sous la forme d'interaction entre objets
  2. de décrire l'état des objets de façon abstraite en termes de réactions vis-à-vis de leur environnement et des messages qui leur sont envoyés.

Exemple : diagramme de communication

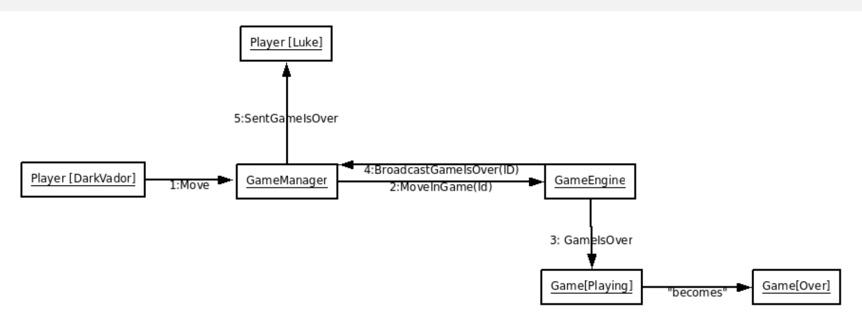


Figure: Diagramme de communication (ou “collaboration”, dans UML 1.x).

- On représente ici un scénario comme un enchaînement d’envoi de message entre objets
- Les objets soulignés correspondent à des instances.
  - Le nom entre crochet symbolise l’état de l’objet à chaque étape du scénario (lorsqu’il est informatif).
- La séquence d’échange des messages est ordonné en utilisant des indexes numériques.

Exemple : diagramme de communication (cont.)

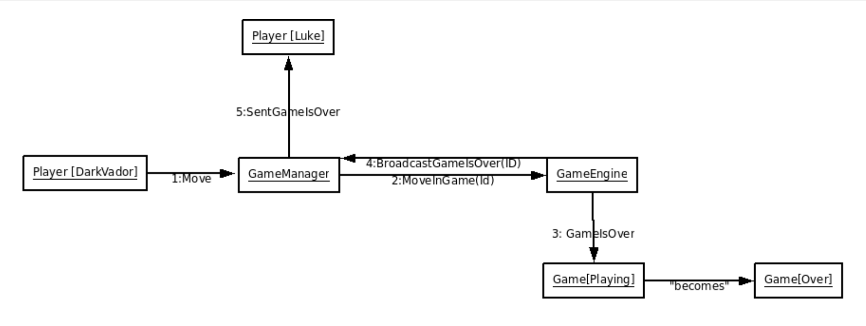


Figure: Diagramme de communication (ou “collaboration”, dans UML 1.x).

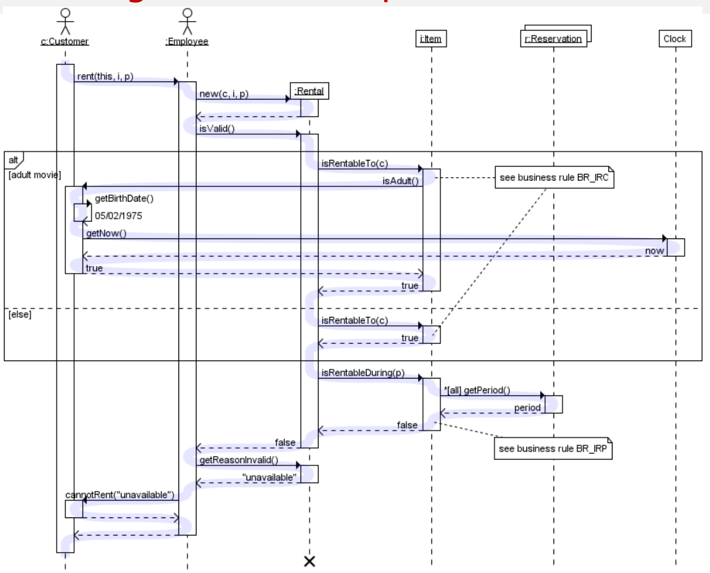
Critique :

- Cette notation met l’accent sur les objets nécessaires à la réalisation d’un cas d’utilisation.
- Il peut être difficile à lire, car les messages sont éparpillés sur le diagramme (voir : le diagrammes de séquence).

Diagramme de séquence

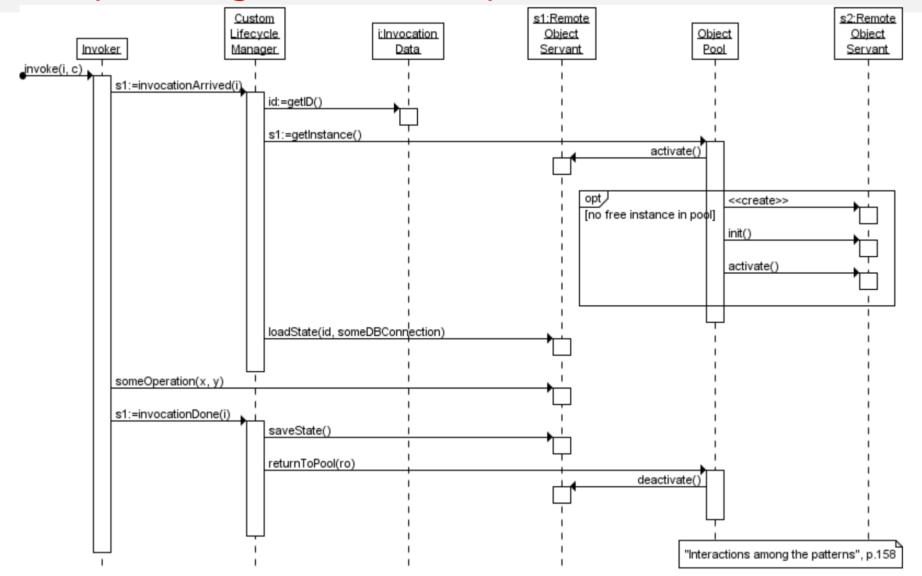
- Un diagramme de séquence présente les interactions entre les objets comme une succession de message
- On peut y dénoter des contraintes de réponses synchrones ou asynchrones, des états bloquants, des protocoles requête/réponse ou *fire and forget*, . . .
- La ligne du temps est bien définie sur l'axe verticale du diagramme
- Cette notation met l'accent sur le protocole de communication entre les objets.

Exemple : diagramme de séquence



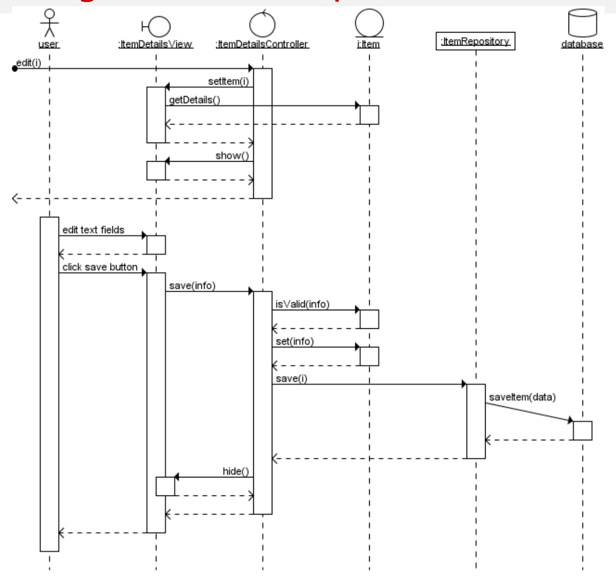
http://www.tracemodeler.com/gallery/

Exemple : diagramme de séquence (cont.)



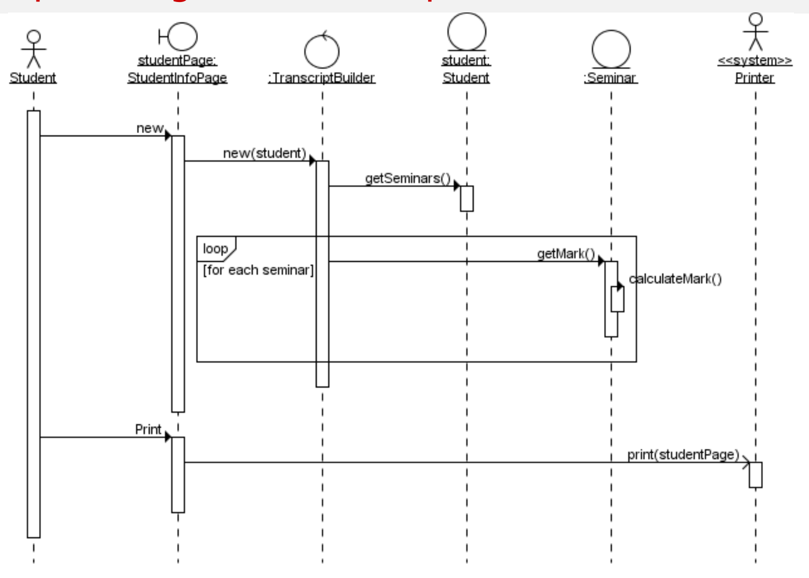
http://www.tracemodeler.com/gallery/

Exemple : diagramme de séquence (cont.)



http://www.tracemodeler.com/gallery/

Exemple : diagramme de séquence (cont.)



http://www.tracemodeler.com/gallery/

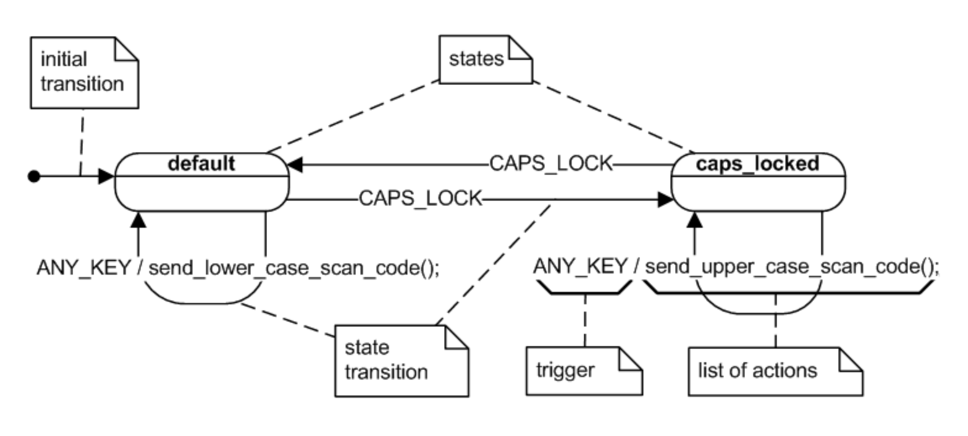
Diagramme d’état

- Les diagrammes d’état (*UML state machine* ou *UML statechart*) représentent l’évolution de l’état du système (ou d’un sous-système) sous la forme d’un automate.
- Une transition de cet automate est suivie en réaction à un événement.
- Elle peut être conditionnée par des contraintes exprimées sur le système.

Intuition

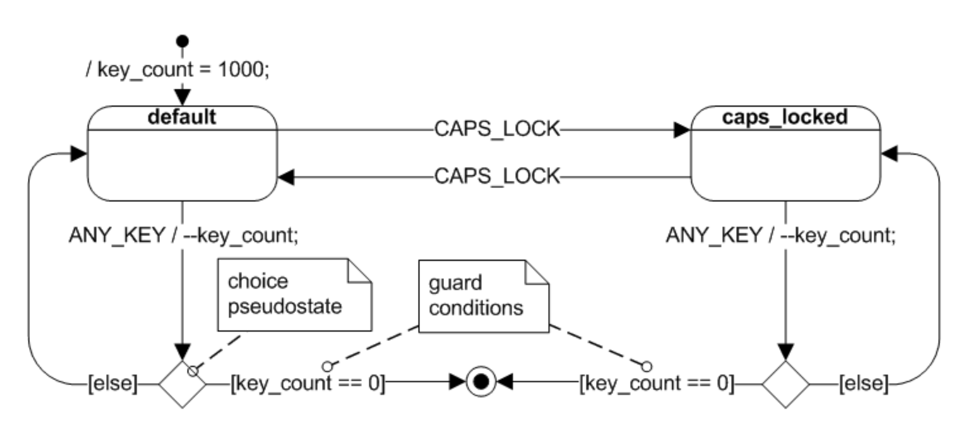
UML statechart	$\simeq$	automate fini	
	+	héritage entre les états	
	+	machine de Mealy	(sortie → état)
	+	machine de Moore	(sortie → état + trans.)
	+	variables & gardes	
	+	...	

Exemple : diagramme d'état



http://en.wikipedia.org/wiki/File:UML_state_machine_Fig1.png

Exemple : diagramme d’état (cont.)



http://en.wikipedia.org/wiki/File:UML_state_machine_Fig2.png

Diagramme d'activité

- un diagramme d'activité modèle un processus business (ou *work-flow*), ses choix, et comment ils interagissent avec le contexte dans lequel le système sera déployé
  - plusieurs systèmes et sous-systèmes, faisant partie ou pas du système en cours de développement, peuvent apparaître dans un diagramme d'activité
- les diagrammes d'activité permettent d'exprimer choix, concurrence/synchronisation, et itérations
- une sémantique (en termes de réseaux de Petri) a été proposée pour les diagrammes d'activité
- originalement encodés comme diagrammes d'état dans UML 1.x, ont été séparés à partir de UML 2.x

“Grammaire” des diagrammes d’activité

rectangles arrondis activités

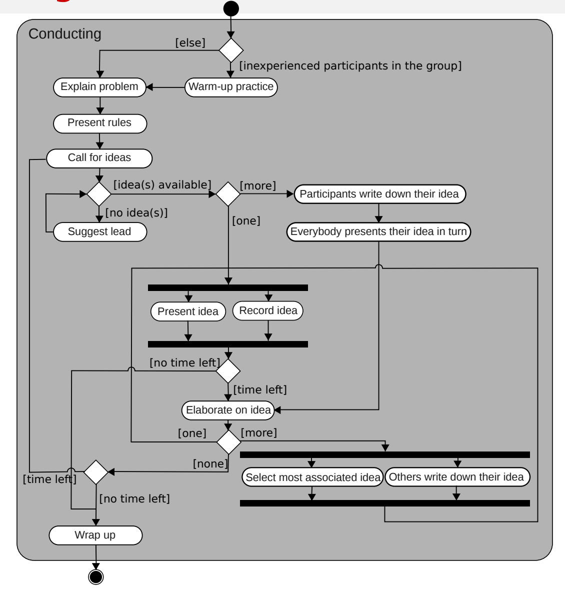
losanges choix/décisions

barres concurrence (*fork*) et synchronisation (*join*)

circles noirs état initial

circles noirs contournés état final

Exemple : diagramme d'activité



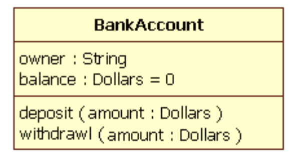
http://en.wikipedia.org/wiki/File:Activity_conducting.svg

Sommaire

Vues statiques

- Les vues statiques établissent la structure du système, à un niveau de détail plus fin que celui du diagramme de paquets.
- Il s'agit d'énumérer les différentes classes d'objets et leur relations.

Exemple : diagramme de (une) classe



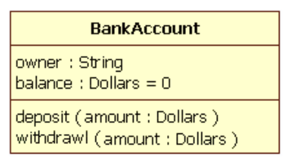
http://en.wikipedia.org/wiki/File:BankAccount.jpg

Figure: Diagramme de classe, réduit à une classe

on retrouve, pour chaque classe :

1. nom de la classe (unique dans le paquet)
2. attribues avec types (et valeur initial)
3. méthodes avec noms et types (d'entrée et sortie)

Exemple : diagramme de (une) classe



http://en.wikipedia.org/wiki/File:BankAccount.jpg

Figure: Diagramme de classe, réduit à une classe

méthodes et attribues peuvent être annotés avec leur visibilité (*scope*) ; pour cela, UML offre des préfixes standardisés :

- \+ public (*default*)
- \# protected
- \- private
- ~ package

Relation entre classes

Les classes peuvent être mise en relation.

UML propose les relations suivantes :

Association un lien sémantique entre deux classes.

Agrégation/composition une relation d'appartenance.

Généralisation/spécialisation une relation d'abstraction.

Instanciation une relation d'affectation de paramètres.

Réalisation une relation de conformité entre une interface et une implémentation.

Relation d'association

- Il s'agit de la notion mathématique de relation (*has a*).
- Une relation a une arité à gauche et à droite.
- Chaque objet impliqué a un rôle dans la relation.

Exemples

- Un scénario est joué par un joueur dans un partie.
- Une action est applicable sur plusieurs objets d'une scène.
- Des objets sont nécessaires pour autoriser une action.

Exemple : diagramme de classe avec association

Student
Name
Address
Phone Number
Email Address
Student Number
Average Mark
Is Eligible To Enroll
Get Seminars Taken

1 enrolled 1..*

Enrollment
Marks Received
Get Average To Date
Get Final Mark

1..* in 1

Seminar
Name
Seminar Number
Fees
Add Student
Drop Student

0..* {ordered, FIFO} on waiting list 0..*

Professor
Name
Address
Phone Number
Email Address
Salary

0..1 instructs 0..*

?Some seminars may not have an instructor?

Un association est formée par :

- un nom ;
- des multiplicités à gauche et à droite ;
- des rôles affectés à chaque objet.



Syntaxe des multiplicités

Un entier "n" *n* objets interviennent dans la relation.
L'étoile "*" plusieurs objets interviennent.
Le segment "`n..*`" au moins *n* objets interviennent.
Le segment "`n..m`" au moins *n* et au plus *m* objets interviennent.

Agrégation/Composition

- L'agrégation est une relation d'appartenance
  - L'agrégation est forcement binaire (l'association non)
  - Exemples (*container*) :
    - Les pièces d'un échiquier lui appartiennent.
    - Les joueurs d'une partie appartiennent à la partie.
- La composition est une relation d'agrégation qui établit une relation de vie ou de mort d'un objet sur un autre.
  - Exemples
    - Si l'échiquier est détruit alors ses pièces aussi.
    - Si une partie est terminée, les joueurs peuvent en jouer une autre. (Ils survivent à la partie.)

Note

La composition est une relation assez subtile et souvent tributaire de certains choix d'implémentation. Il est souvent préférable de ne pas l'utiliser (sauf en C++ car la gestion explicite de la mémoire nécessite une réflexion précise sur la notion de durée de vie qu'il faut considérer dès la phase de spécification).

Exemple : compositions et agrégations

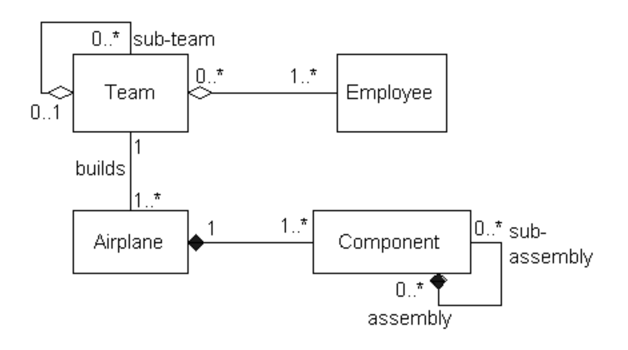


Figure: diagramme de classe avec compositions et agrégations.

- Le losange vide signifie «est agrégé à».
- Le losange plein signifie «est composé de».

Objectifs de la généralisation/spécialisation

Spécialisation

- Ajout d'une fonctionnalité.
- Focalisation sur un aspect spécifique à une classe.

Généralisation

- Factorisation de critères communs.
- Abstraction des détails.

Analogie avec la relation d'inclusion ensembliste (*is a*).

Définition (Classe abstraite)

Une classe est abstraite si elle n'est jamais vouée à être instanciée.

Être abstraite capture la notion de "concept".

Exemple : généralisation

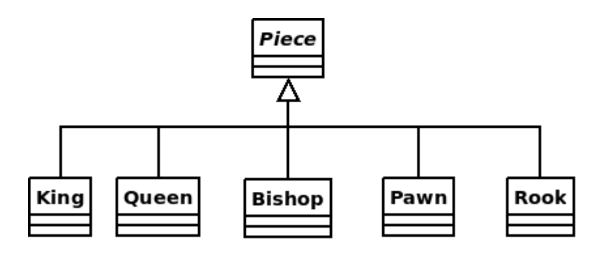


Figure: Diagramme de classe (avec relation de généralisation).

- «Piece» est une super-classe de «Queen», elle généralise cette dernière.
- «Queen» est une sous-classe de «Piece», elle spécialise cette dernière.

On exprime ici une relation d'abstraction entre composants.

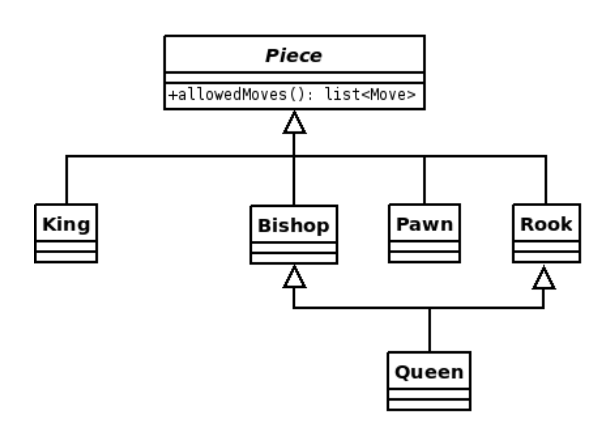
Annotations de la relation de généralisation

On peut annoter la relation de généralisation par :

- **complete** on ne peut plus rajouter une nouvelle sous-classe.
  incomplete on pourra rajouter une nouvelle sous-classe dans le futur.
- **disjoint** les sous-classes ne pourront pas être les parents d'une future sous-classes.
  overlap les sous-classes pourront être utilisées comme super-classes d'une même sous-classe dans le futur.

Intuition : complete/incomplete s'occupent de la "extensibilité horizontale" de la généralisation ; disjoint/overlap de sa "extensibilité verticale"

Exemple : mauvaise généralisation



- La relation suivante n'est pas correcte puisqu'une règle n'est pas un cas particulier (*is a*) de tour et de fou.
- Il ne faut pas confondre factorisation de code et généralisation.

Exemple : diagramme composite

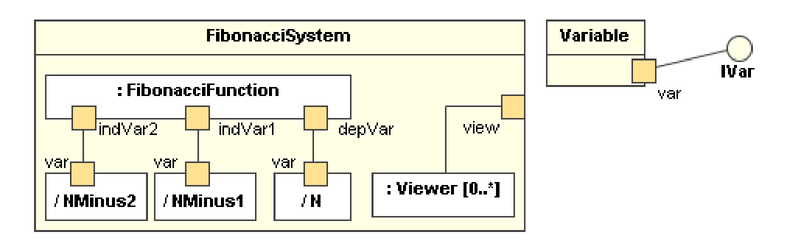


Figure: Diagramme composite.

- Les diagrammes composites servent à donner une vision abstraite de la structure interne d'une classe.
- On brise ici le principe d'encapsulation : à n'utiliser qu'en cas de stricte nécessité.

Exemple : diagramme de réalisation

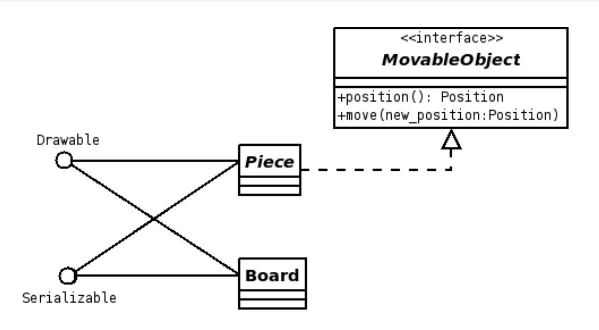


Figure: Diagramme de réalisation.

- Un diagramme de réalisation illustre la compatibilité entre un objet et une interface.
- Il y a deux notations possibles pour cela en UML :
  1. Un lien vers un cercle faisant référence au nom de l'interface.
  2. Une relation de généralisation en pointillés vers une description précise de l'interface.

Exemple : diagramme de rôle

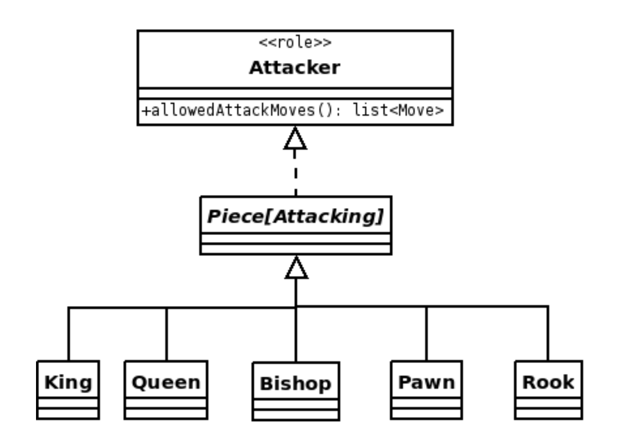


Figure: Diagramme de rôle ou de profil.

- Un rôle peut être joué par un objet dans une situation particulière ou en fonction de son état.

Exemple : mauvais rôle

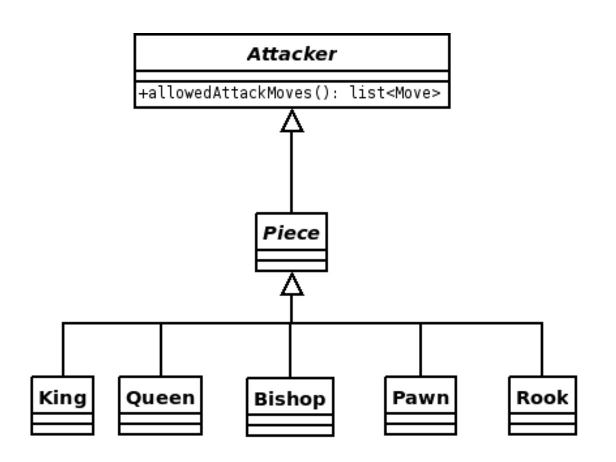


Figure: Point de vue incorrect sur un rôle.

- Il faut distinguer généralisation et prise temporaire d'un rôle.

Exemple : bon rôle

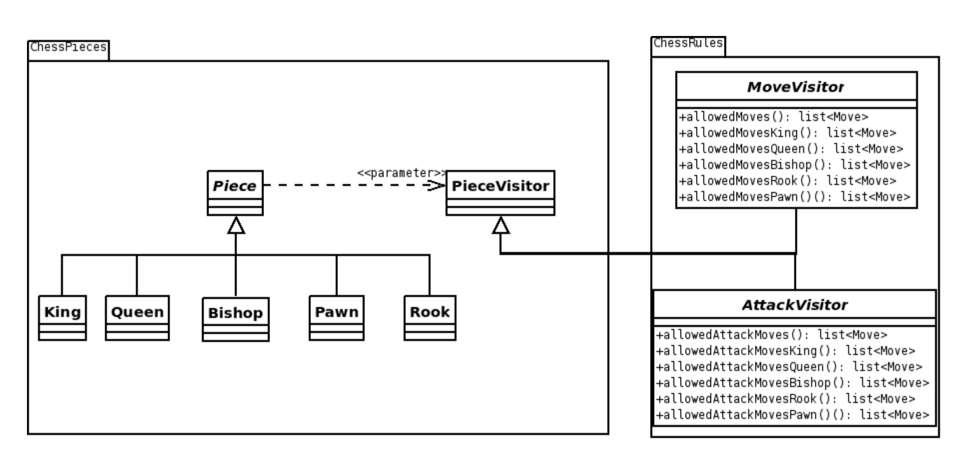


Figure: Une implémentation correcte d'un rôle.

Sommaire

Résumé

- Nous avons brièvement présenté RUP.
  - Nous en étudierons les principes dans le cours de conception orientée objet des systèmes.
- Nous avons survolé UML.
  - Ce sera un outil que nous appliquerons et approfondirons par la suite.